计算机组成

流水线及其冒险

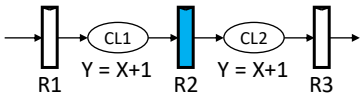
高小鹏

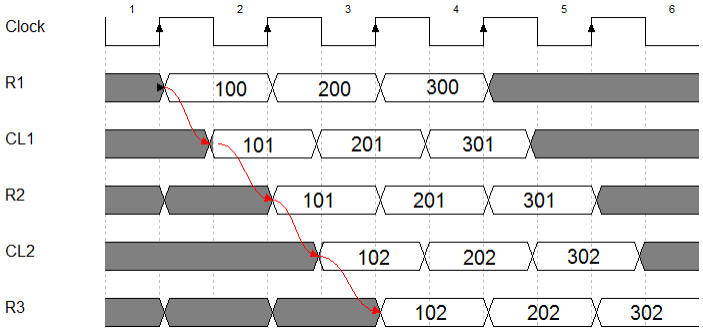
北京航空航天大学计算机学院

目录

示例：简单的流水线电路

- 以下是R1在3个连续时钟周期分别输入100、200、300的电路时序图

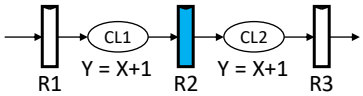




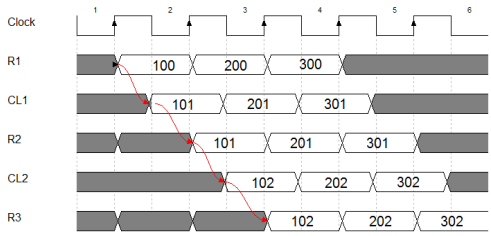
3

流水线工作过程的形式表示

- 为清晰的表示流水线的工作过程，采用表格化的描述方式



- 水平方向：按流水线执行顺序布局
  - 由于寄存器值的保存与传递是分析要点，同时为了简化分析过程，不单独记录组合逻辑计算结果
- 垂直方向：寄存器在相应时钟上升沿后的值



	CL1	CL2	
CLK	R1	R2	R3
↑ 1	XXX→100		
↑ 2	100→200	→101	
↑ 3	100→300	→201	→102
↑ 4		→301	→202
↑ 5			→302

4

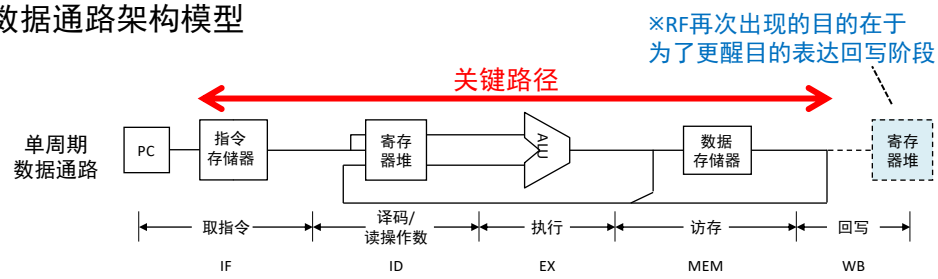
回顾：MIPS数据通路的5个阶段

- 1)取指令：IF，Instruction Fetch
  - 从IM中读取指令
  - 驱动NPC更新PC
- 2)译码：ID，Instruction Decode
  - 指令驱动RF读取寄存器值(包括立即数扩展)
  - 控制器对指令的op和funct进行译码
- 3)执行：EX，EXecution
  - ALU计算（load/store计算地址；其他指令则为计算）
- 4)访存：MEM，MEMory
  - 读(或写)存储器
- 5)回写：WB，Write Back
  - 将ALU结果或存储器读出的数据写回寄存器堆

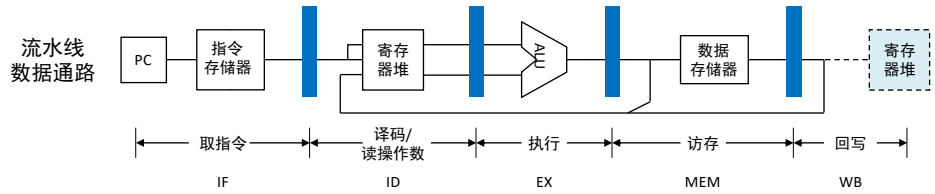
5

流水线架构模型

- 单周期数据通路架构模型



- 流水线架通过插入多个寄存器，从而在物理上将单周期的关键路径切割为若干段



7

流水线执行特点

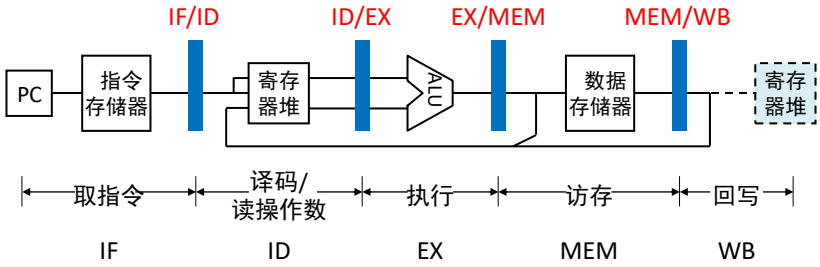
- 理论上，流水线每个cycle都从IM中读取一条新指令，同时将已在流水线里的指令向前推进一个阶段
  - 多条指令在数据通路中，但同一时刻占用不同的资源（即处于不同阶段）

- 对于N级流水线，单条指令执行周期数均为N
  - 少数控制流指令除外（如分支类指令、跳转类指令）
- 理论上，当流水线充满后，每个cycle可以执行完一条指令

8

流水线寄存器命名

- 方法1：采用相邻两级名字组合的方式

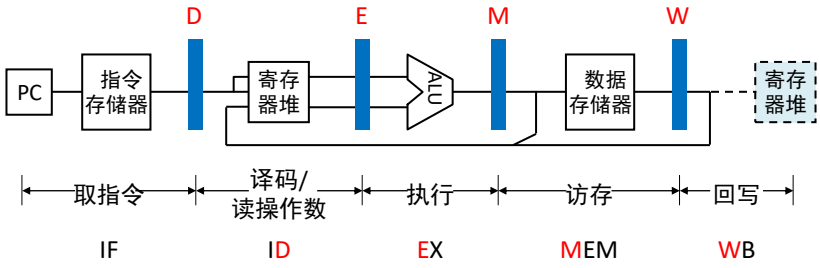


- 优点：各级寄存器的位置非常清晰
- 缺点：当需要标记位于各级寄存器的指令某些信息时，名字会非常的长

9

流水线寄存器命名

- 方法2：用各级功能的某个字母命名



- 优点：当需要标记位于各级寄存器的指令某些信息时，名字非常简洁

时钟驱动的流水线时空图

- 用于精确分析指令/周期/寄存器3者关系
  - 指令何时处于何阶段
- 理想情况下，在clk5后流水线全部充满
  - 所有部件都在执行指令：不同指令位于不同部件

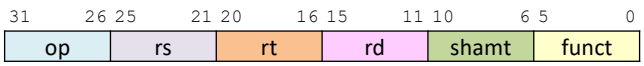
				IF级	ID级	EX级	MEM级	WB级	
相对PC的地址偏移	指令	CLK	PC	IM	D	E	M	W	RF
0	Instr 1	↑ 1	0→4	Instr 1	Instr 1				
4	Instr 2	↑ 2	4→8	Instr 2	Instr 2	Instr 1			
8	Instr 3	↑ 3	12→12	Instr 3	Instr 3	Instr 2	Instr 1		
12	Instr 4	↑ 4	12→16	Instr 4	Instr 4	Instr 3	Instr 2	Instr 1	
16	Instr 5	↑ 5	16→20	Instr 5	Instr 5	Instr 4	Instr 3	Instr 2	Instr1
20	Instr 6	↑ 6	16→20	Instr6	Instr6	Instr5	Instr4	Instr3	Instr2

目录

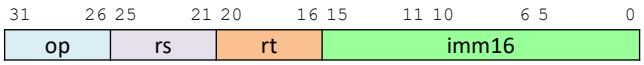
- 流水线概述
- 流水线数据通路
- 流水线控制
- 流水线冒险
- 流水线性能分析
- 2种CPU模型对比

MIPS-C0指令集

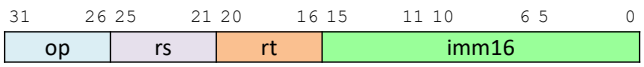
- 加减法指令
  - addu rd,rs,rt
  - subu rd,rs,rt



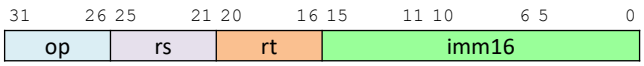
- 立即数指令
  - ori rt,rs,imm16



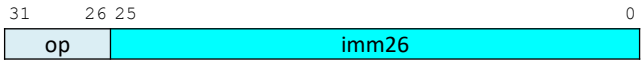
- 存储访问指令
  - lw rt,rs,imm16
  - sw rt,rs,imm16



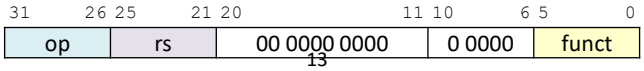
- 分支指令
  - beq rs,rt,imm16



- 函数调用指令
  - jal target

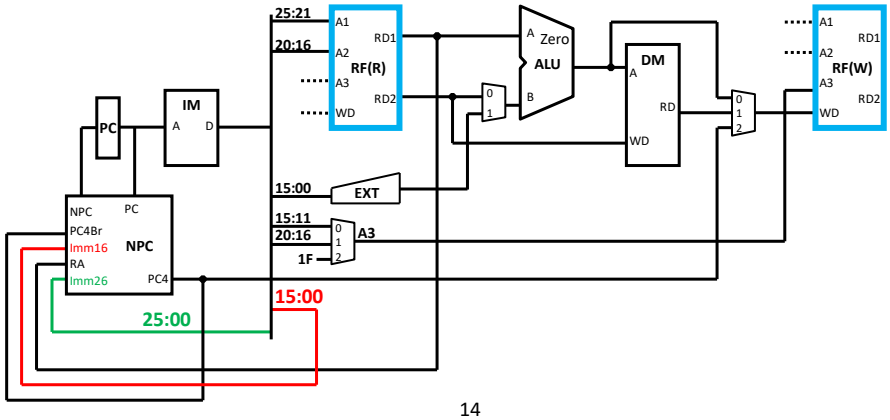


- 函数返回指令
  - jr rs



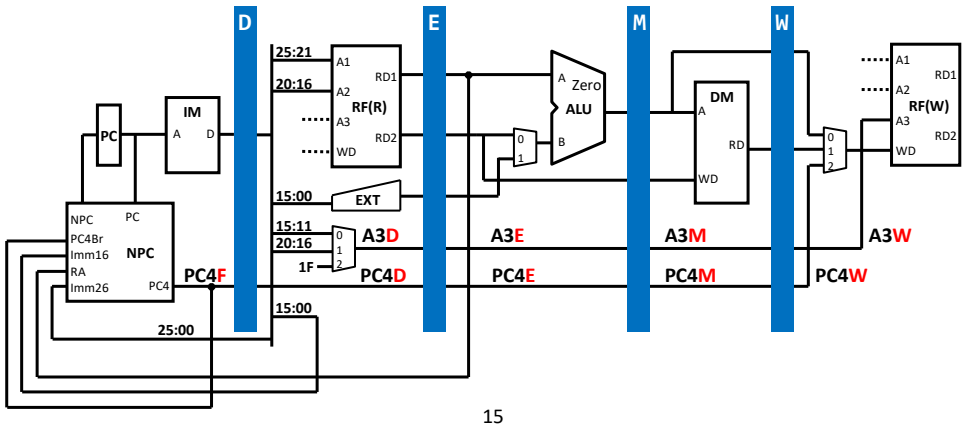
单周期数据通路

- 技术处理1：将RF分为读出和写入两个环节来表示
  - RF(R)代表RF用于读出数据；RF(W)代表RF用于写入数据
- 技术处理2：NPC设置了Imm16和Imm26两个输入，分别对应beq和jal的PC计算需求
  - 不合并两个输入信号，有助于后续的分析



流水线信号命名

- 命名规范：信号名的末尾添加特定字母
  - 这种命名方式既保留了信号的基本语义，又清晰的区分了信号所在阶段
- 示例：1）NPC输出的PC4；2）指令[20:16]与[15:11]产生的目的寄存器编号A3

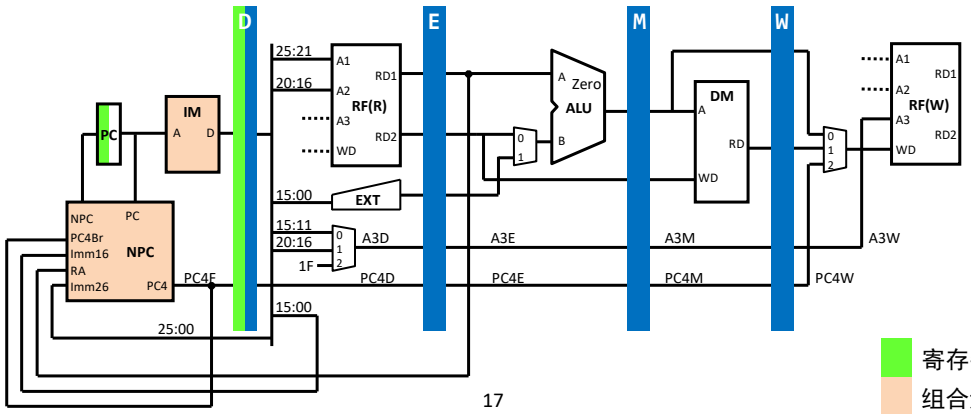


目录

示例：lw执行过程1/5

- 第1个↴后
  - PC驱动IM读取指令，lw写入D
  - PC驱动NPC计算，将PC+4写入PC

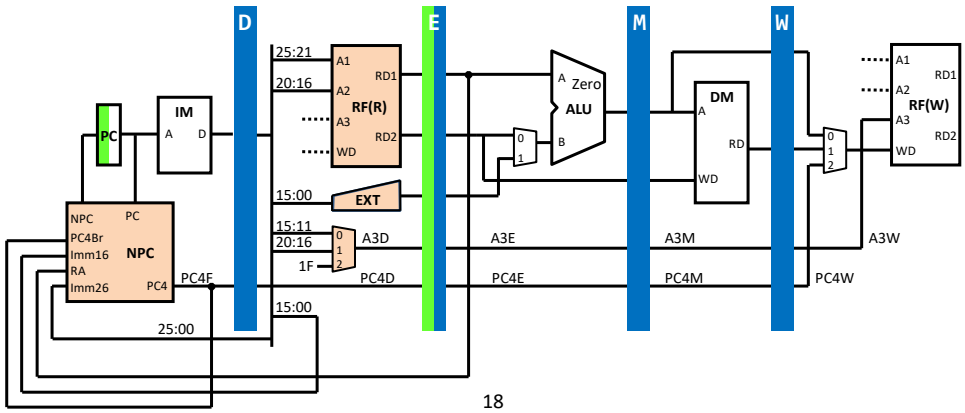
地址	指令	CLK	PC	D	E	M	W	RF
00	lw	↴1	00→04	→lw				



示例：lw执行过程2/5

- 第2个↑后
  - D驱动RF读取操作数，结果写入E
  - D驱动EXT计算扩展的立即数，结果写入写入E

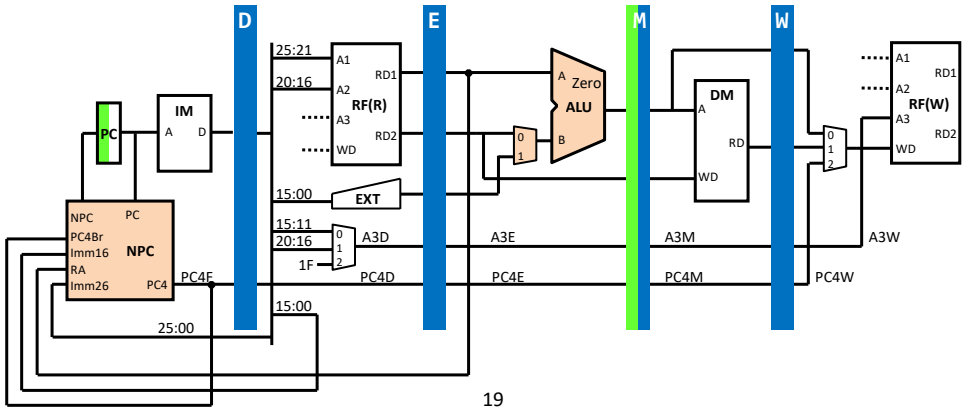
地址	指令	CLK	PC	D	E	M	W	RF
00	lw	↑1	00→04	→lw				
04		↑2	04→08		→lw			



示例：lw执行过程3/5

- 第3个↑后
  - E驱动ALU计算，结果写入M

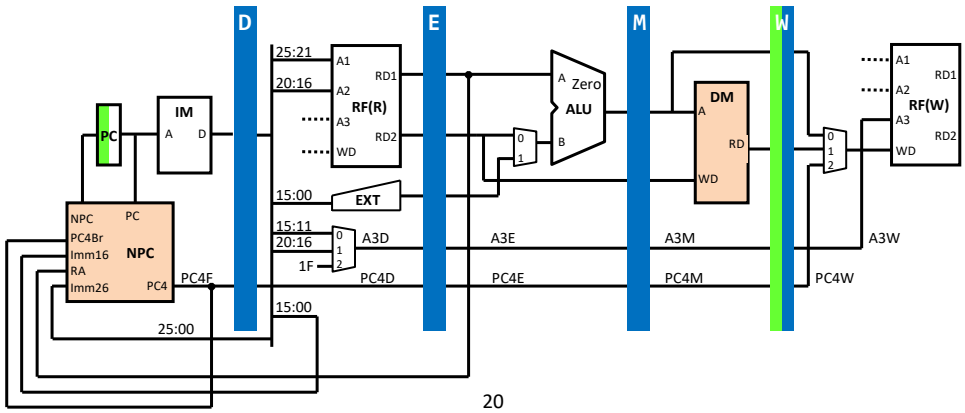
地址	指令	CLK	PC	D	E	M	W	RF
00	lw	↑1	00→04	→lw				
04		↑2	04→08		→lw			
08		↑3	08→12			→lw		



示例：lw执行过程4/5

- 第4个↑后
  - M驱动DM读取数据，结果写入W

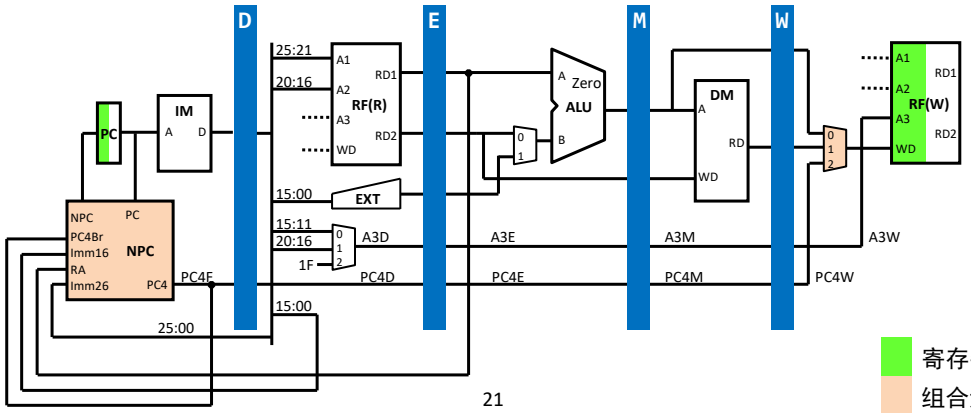
地址	指令	CLK	PC	D	E	M	W	RF
00	lw	↑1	00→04	→lw				
04		↑2	04→08		→lw			
08		↑3	08→12			→lw		
12		↑4	12→16				→lw	



示例：lw执行过程5/5

- 第5个↑后
  - W将结果写入RF

地址	CLK	PC	D	E	M	W	RF
00	↑1	00→04	lw				
04	↑2	04→08	I2	lw			
08	↑3	08→12	I3		→lw		
12	↑4	12→16	I4			→lw	
16	↑5	16→20	I5				更新



目录

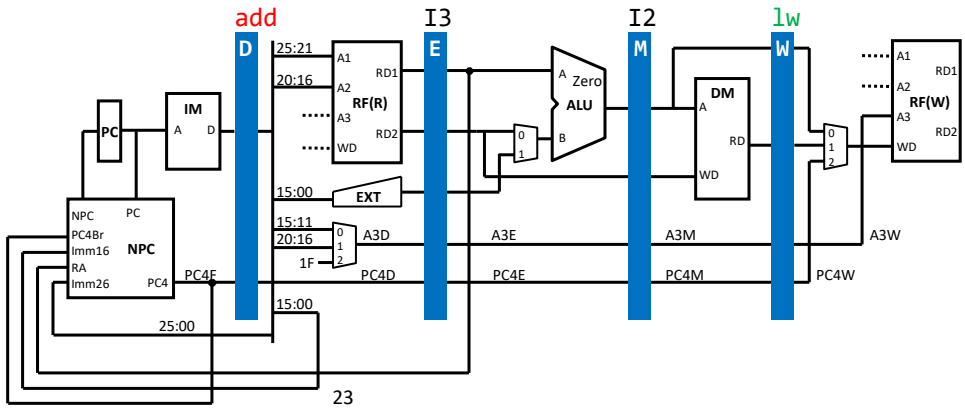
- 流水线概述
- 流水线数据通路
  - D级能否读出W级写入RF的值？
- 流水线冒险
- 流水线性能分析
- 2种CPU模型对比

程序员视角的指令执行结果

- 考察如下指令序列
  - 假设1：$1初值为7，lw执行后$1值为8
  - 假设2：除lw外，其他指令均与$1无关
- 从程序员视角，add与sub读取$1时，其结果均应为8

```
 0   lw $1, 0(XX)
 4   I2
 8   I3
12   add XX, $1, XX
16   sub XX, $1, XX
20   I6
```

※$1的初值为7
※lw执行后$1为8



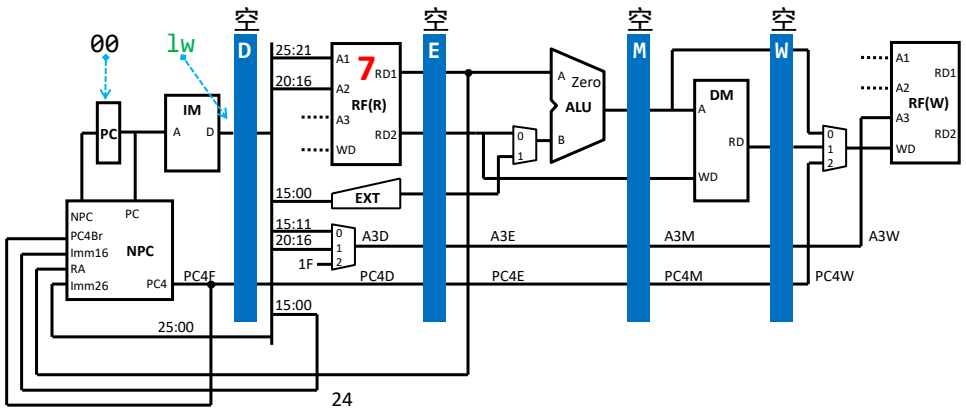
第1个↴前的初始状态

- 1）PC为00，指向lw
- 2）IM输出指令lw
- 3）各级流水寄存器均为空
- 4）$1的值为7

CLK	PC	D	E	M	W	$1
	00					7

```
 0   lw $1, 0(XX)
 4   I2
 8   I3
 12  add XX, $1, XX
 16  sub XX, $1, XX
 20  I6
```

※假设$1的初值为7
※假设lw执行后$1为8



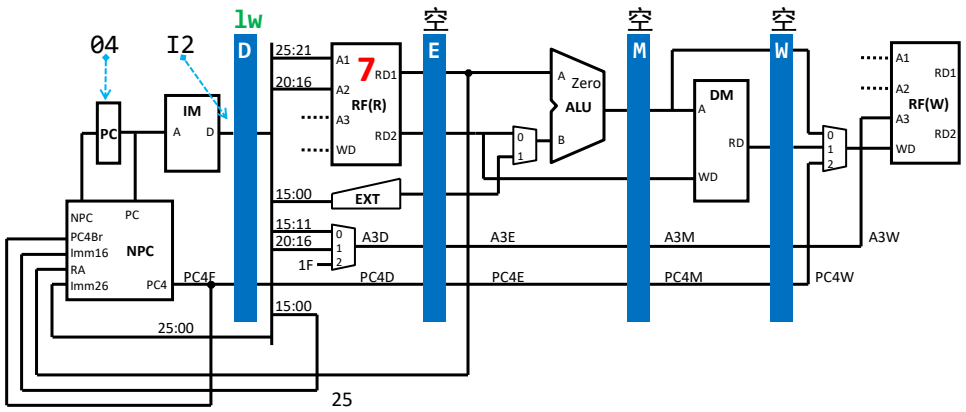
第1个↴后

- 1）PC自增变为04
- 2）IM输出指令I2
- 3）lw进入D级

CLK	PC	D	E	M	W	$1
↴1	00→04	lw				7
↴2						
↴3						
↴4						
↴5						

```
 0   lw $1, 0(XX)
 4   I2
 8   I3
 12  add XX, $1, XX
 16  sub XX, $1, XX
 20  I6
```

※假设$1的初值为7
※假设lw执行后$1为8



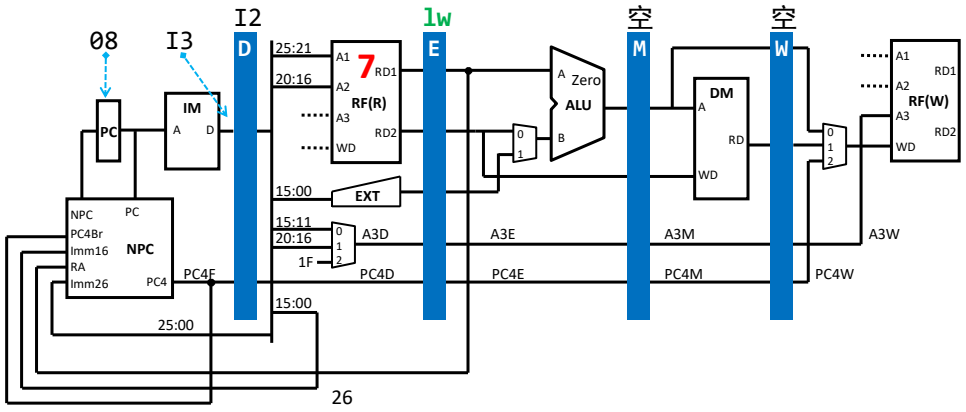
第2个↴后

- 1）PC变为08
- 2）IM输出指令I3
- 3）lw进入E级，I2进入D

CLK	PC	D	E	M	W	$1
↴1	00→04	lw				7
↴2	04→08	I2	lw			7
↴3						
↴4						
↴5						

```
0   lw $1, 0(XX)
4   I2
8   I3
12  add XX, $1, XX
16  sub XX, $1, XX
20  I6
```

※假设$1的初值为7
※假设lw执行后$1为8



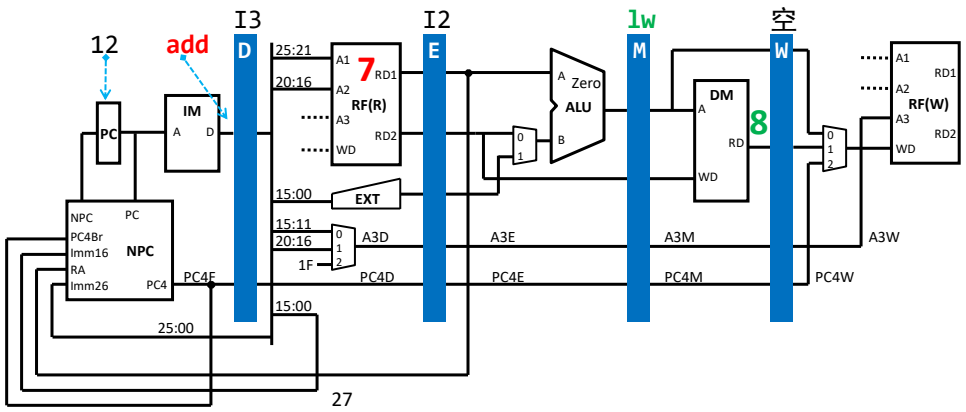
第3个↴后

- 1）PC变为12
- 2）IM输出指令add
- 3）lw进入M级，I3进入D
  - 由于lw进入M后就开始读取DM，因此DM的输出值为8

CLK	PC	D	E	M	W	$1
↴1	00→04	lw				7
↴2	04→08	I2	lw			7
↴3	08→12	I3	I2	lw		7
↴4						
↴5						

```
0   lw $1, 0(XX)
4   I2
8   I3
12  add XX, $1, XX
16  sub XX, $1, XX
20  I6
```

※假设$1的初值为7
※假设lw执行后$1为8



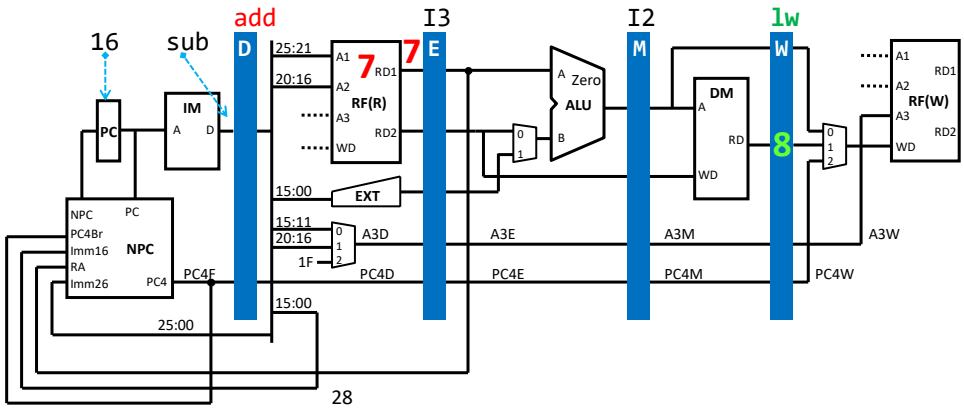
第4个↴后

- 1）PC变为16
- 2）IM输出指令sub
- 3）lw进入W级，add进入D
  - 由于add进入D后就开始读取RF，因此RF输出的$1值为7

CLK	PC	D	E	M	W	$1
↴1	00→04	lw				7
↴2	04→08	I2	lw			7
↴3	08→12	I3	I2	lw		7
↴4	12→16	7@add	I3	I2	8@lw	7
↴5						

```
0   lw $1, 0(XX)
4   I2
8   I3
12  add XX, $1, XX
16  sub XX, $1, XX
20  I6
```

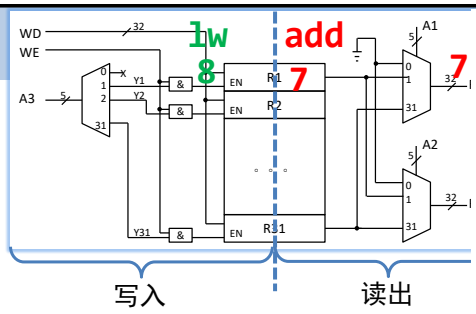
※假设$1的初值为7
※假设lw执行后$1为8



28

第4个↴后，第5个↴前

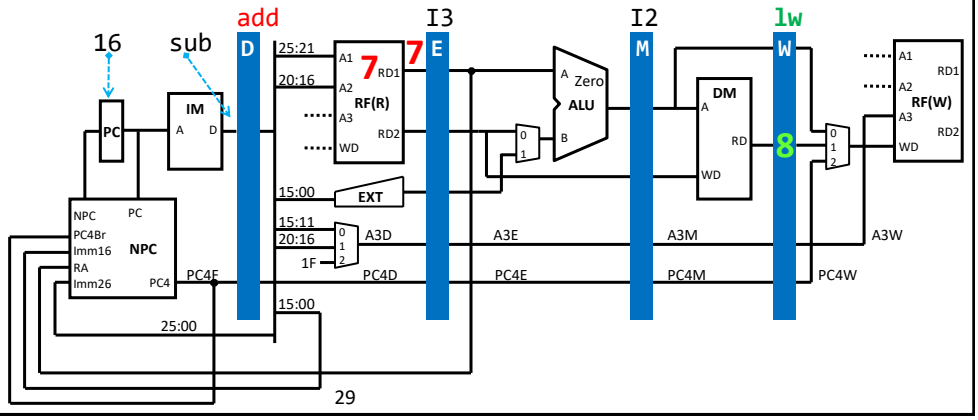
- 观察RF内部
  - 1）$1的D输入为lw的回写数据，即8
  - 2）由于↴没来到，因此$1尚未被写入，add读出的$1仍然输出7



CLK	PC	D	E	M	W	$1
↴1	00→04	lw				7
↴2	04→08	I2	lw			7
↴3	08→12	I3	I2	lw		7
↴4	12→16	7@add	I3	I2	8@lw	7
↴5						

```
0   lw $1, 0(XX)
4   I2
8   I3
12  add XX, $1, XX
16  sub XX, $1, XX
20  I6
```

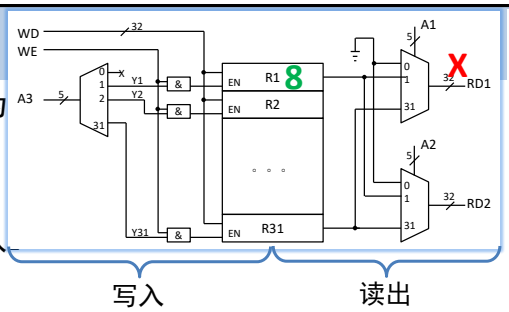
※假设$1的初值为7
※假设lw执行后$1为8



29

第5个↴后

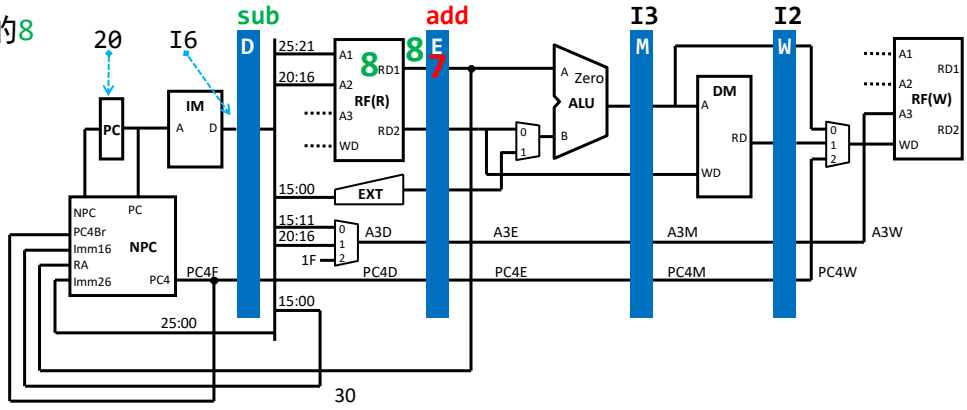
- ↴到来时，3个行为同时发生：
  - 1）$1写入8
  - 2）add读出的7写入
  - 3）sub进入D
- sub读出的是正确的8



CLK	PC	D	E	M	W	$1
↴1	00→04	lw				7
↴2	04→08	I2	lw			7
↴3	08→12	I3	I2	lw		7
↴4	12→16	7@add	I3	I2	8@lw	7
↴5	16→20	8@sub	7@add	I3	I2	8

```
0   lw $1, 0(XX)
4   I2
8   I3
12  add XX, $1, XX
16  sub XX, $1, XX
20  I6
```

※假设$1的初值为7
※假设lw执行后$1为8

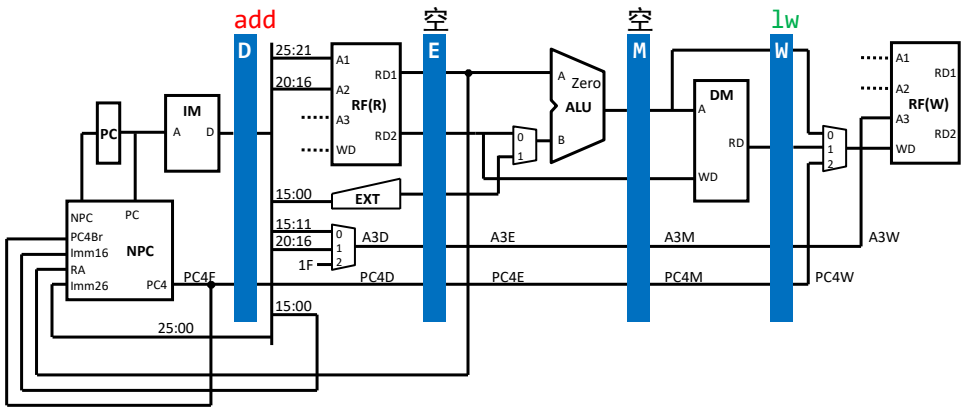


结论：同时读写同一个寄存器时，会产生一致性问题

- 导致错误的根本原因在于RF的寄存器行为特征
  - 1）寄存器的值只有在↴时才会改变
  - 2）寄存器的输出值在↴时同时也写入了下一级寄存器
- 解决问题的方法：转发（在数据冒险部分介绍）

```
0   lw $1, 0(XX)
4   I2
8   I3
12  add XX, $1, XX
16  sub XX, $1, XX
20  I6
```

※假设$1的初值为7
※假设lw执行后$1为8



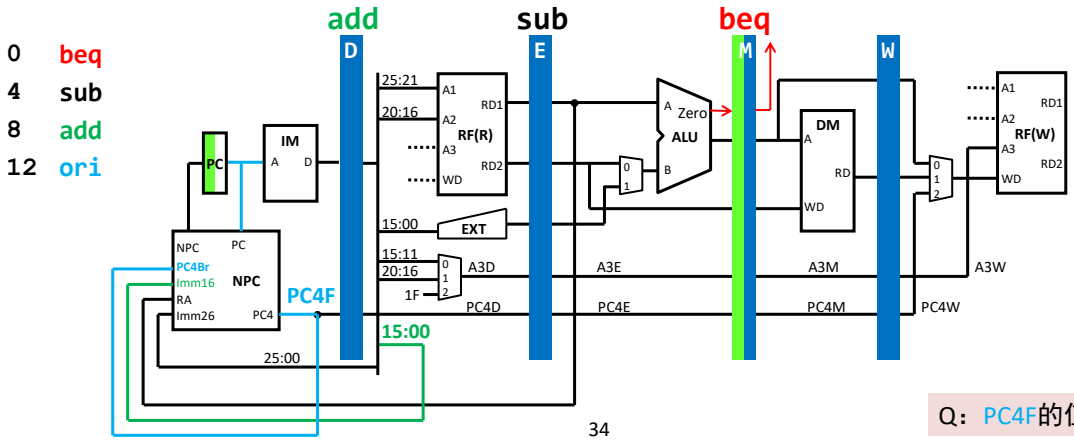
目录

33

示例：Beq错误的计算地址

- 第3个↴后
  - beq进入M级，add进入D级，PC变为12
- NPC计算地址时会用到3种完全无关的信息
  - beq的Zero、add的偏移、ori指令的PC4F

地址	指令	CLK	PC	D	E	M	W	RF
00	beq	↴1	00→04	beq				
04	sub	↴2	04→08	sub	beq			
08	add	↴3	08→12	add	sub	beq		

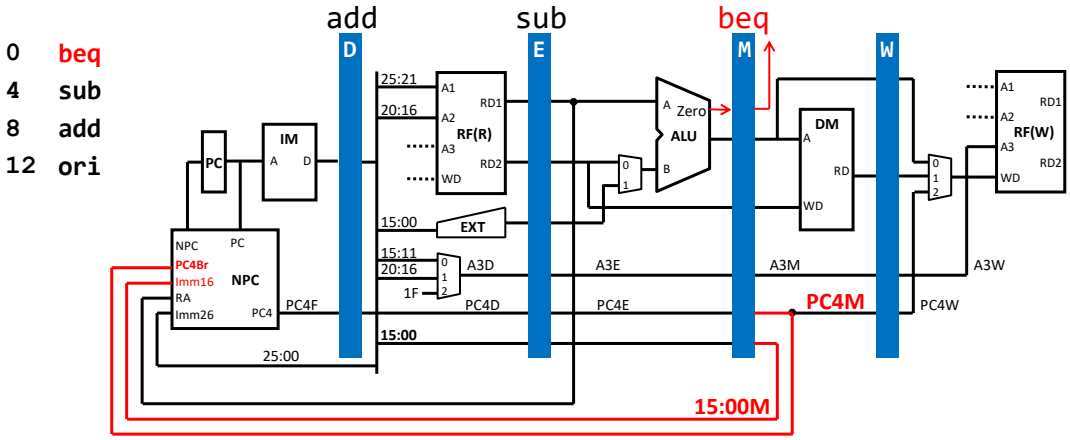


Q：PC4F的值是多少？

34

确保同步：修正beq的数据通路

- 出错原因：指令中同一功能的多个相关信息没有同步传递
- 为了确保beq正确计算地址，必须做到2点
  - 1）beq的所有相关信息（包括16位偏移）必须同步传递
  - 2）必须使用同级的相关信息计算地址



目录

- 流水线概述
- 流水线数据通路
  - 性能
- 流水线控制
- 流水线冒险
- 流水线性能分析
- 2种CPU模型对比

流水线性能

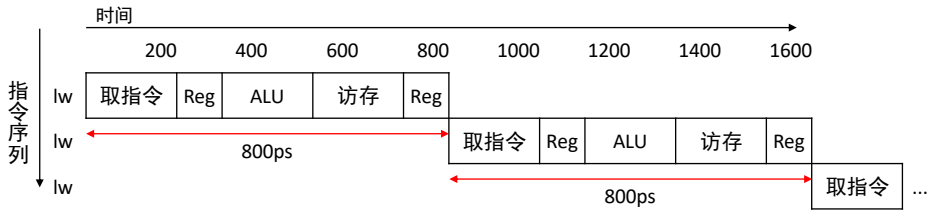
- 假设采用单周期时的性能参数，流水线的时钟频率是多少？
  - 单周期：lw是关键路径，延迟为800ps，因此时钟频率上限为1.25GHz
  - 流水线：各段最大延迟为200ps，即因此时钟频率上限为5GHz

指令	读取指令	读寄存器	ALU	数据存取	写寄存器
addu	200	100	200		100
subu	200	100	200		100
ori	200	100	200		100
lw	200	100	200	200	100
sw	200	100	200	200	
beq	200	100	200		
jal	200				100
jr	200	100			

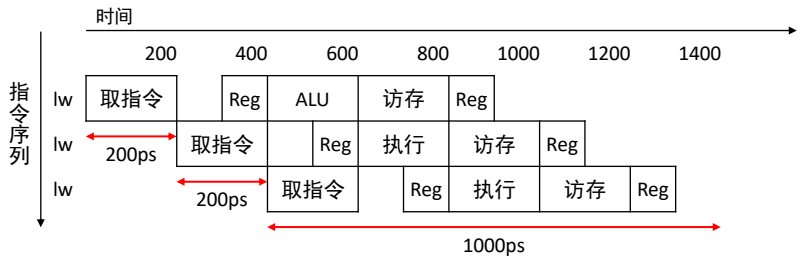
37

流水线性能

- 单周期：每个时钟周期800ps，主频1.25GHz



- 流水线：每个时钟周期200ps，主频5GHz



38

指令级并行

- 流水线使得CPU可以在同一时刻执行多条指令
  - 多条指令同时运行
  - 各功能部件利用率高
  - 大幅度提高了CPU的吞吐率
- 这种技术被称为指令级并行
  - ILP：Instruction Level Parallelism

39

流水线的基本特性

$$\text{流水线加速比} = \frac{\text{单周期的程序执行时间}}{\text{流水线的程序执行时间}}$$

- 理想加速比＝流水线级数
  - 假设单周期数据通路均匀切分为N段，则流水线性能是单周期的N倍！
- 如果各流水段执行时间不平衡，则加速比下降
- 填充流水线和排放流水线，导致加速比下降
- 流水线不改善单条指令执行周期数
- 流水线改善的是吞吐率
  - 理论上，当流水线充满后，每个时钟周期可以执行完一条指令
- 流水线时钟频率受限于最慢的流水段

40

指令集对流水线设计的影响

- MIPS指令集是面向流水线架构设计的
- 所有的指令都是32位
  - 取指和译码都能在一个周期内完成
- 指令格式种类少且规整，2个源寄存器的位置保持不变
  - 这使得读取操作数和译码可以同时执行
- 存储器操作只有load和store
  - 第3拍计算地址，第4拍访存
- 存储器操作是地址对齐的
  - 有利于与主存的协同设计，且访存周期数固定

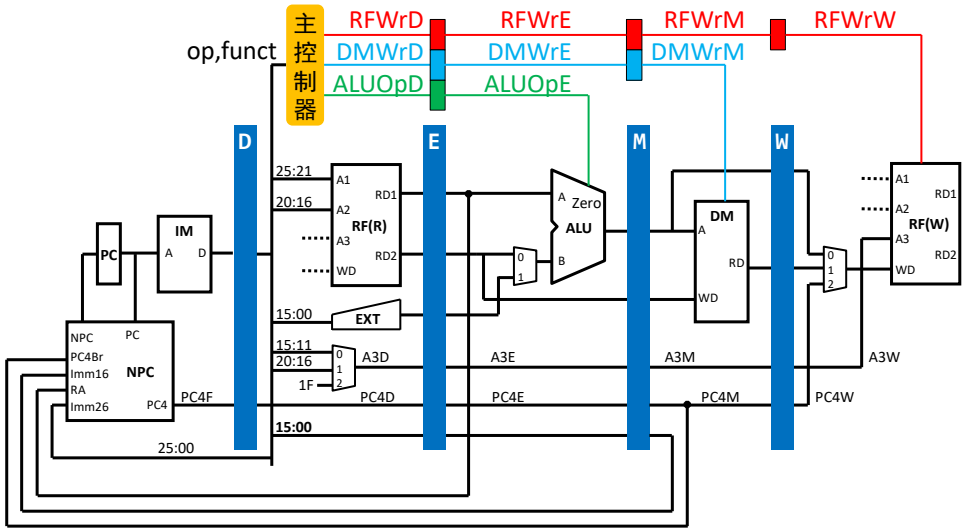
41

目录

42

流水的主控制信号

- 与寄存器值与编号类似，主控制信号也是指令的一部分，必须同步传递
- 主控制信号产生方式与单周期完全相同



控制信号的额外含义

- 单周期
  - 控制信号只是用于控制功能部件执行正确的功能
  - 示例：RFWr有效意味着可以写入RF
- 流水线
  - 除了具有单周期的相同意义外，控制信号在各级寄存器中流水传递，还代表了指令执行到哪个阶段
  - 示例：RFWrW有效，表明load类、R型计算类、I型计算类等指令已经进入W级流水线寄存器了，即指令执行到最后一个流水段了

Q：为什么需要了解指令进入到哪个阶段了？

已经在流水线中的多条指令，彼此之间可能会发生冲突。为此，需要了解各条指令在流水线中的所处位置。

目录

45

概述

- 结构冒险
  - 需要的资源被占用
- 数据冒险
  - 指令之间存在数据依赖
  - 后继指令需要等待前序指令执行结束
- 控制冒险
  - 指令流的方向选择依赖于前序指令执行结果

46

目录

47

结构冒险：取指令与访存

- 后继指令需要的资源被前序指令占用，导致资源冲突
- 第3个↴后/第4个↴前：PC驱动IM以读指令；M级lw驱动DM以读数据

相对PC的地址偏移	指令	CLK	IM: PC	D	E	DM: M	W	RF
0	lw	↴ 1	00→04	lw				
4	add	↴ 2	04→08	add	lw			
8	sub	↴ 3	08→12	sub	add	lw		
12	or	↴ 4	12→16	or	sub	add	lw	
16	and	↴ 5	16→20	and	or	sub	add	lw

- 如果IM与DM不分离（即只有一个存储器），则必然导致资源冲突
- 为此，流水线要求数据和指令必须分离存储
  - 这种结构实际上对应了分离的一级指令cache和一级数据cache（相关内容参见后续存储系统）

结构冒险：读寄存器与写寄存器

- 后继指令需要的资源被前序指令占用，导致资源冲突
- 第4个↴后/第5个↴前： D级or驱动RF要读数据；W级lw驱动RF要写数据

相对PC的地址偏移	指令	CLK	IM	RF	ALU	DM	
			PC	D	E	M	W → RF
0	lw	↴ 1	00→04	lw			
4	add	↴ 2	04→08	add	lw		
8	sub	↴ 3	08→12	sub	add	lw	
12	or	↴ 4	12→16	or	sub	add	lw
16	and	↴ 5	16→20	and	or	sub	add

- RF设计：读端口与写端口分离（目前常见的思路）
- 结论：RF不会成为瓶颈

50

目录

基于寄存器的数据相关

- 当2条指令都读写同一个寄存器时，这就产生了数据相关
  - 程序中各语句通过变量完成信息传递
  - 对变量的读写转变为对寄存器的读写
- 读写对寄存器的影响
  - 读(Read)：不会改变寄存器的值
  - 写(Write)：会改变寄存器的值（破坏性操作）
- 对于同一个寄存器，有4种读写组合：R-R，W-W，R-W，W-R
  - 字母先后代表读写的先后顺序
  - 示例
    - lw-sub：在$t0上是W-R
    - add-sub：在$t2上是W-W

```
lw  $t0, 0($t1)
sub $t3, $t0, $t2
and $t5, $t0, $t4
or  $t7, $t0, $t6
add $t1, $t2, $t3
```

```
lw  $t0, 0($t1)     W:写$t0
add $t2, 0($s0)     W:写$t2
sub $t2, $t0, $t4   R:读$t0
                    W:写$t2
```

51

流水线带来的数据一致性问题

- 程序员逻辑：后续指令必须获取到前序指令的结果
  - 1）前序指令未执行，后继指令不执行
  - 2）前序指令执行结束，后继指令才执行
- 单周期：数据通路每次只能执行一条指令
  - 由于数据通路不能同时执行多条指令，因此必然是前序指令全部执行结束后才会执行后继指令
  - CPU执行逻辑与程序员逻辑完全一致
- 流水线：数据通路同时执行多条指令
- 流水线产生一致性问题：后执行的指令是否一定能获取先执行的指令的结果？

52

流水线执行指令与程序正确性

- 案例分析

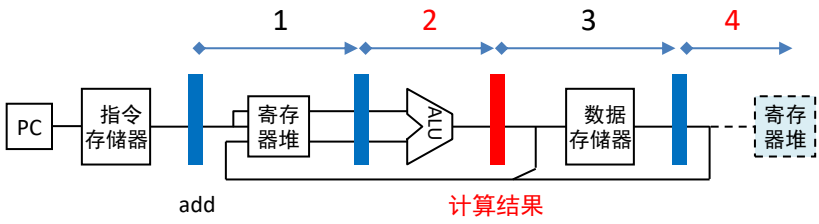
偏移	指令	CLK	PC	D	E	M	W	RF
				IM	RF	ALU	DM	
0	add $1, $2, $3	↑ 1	00→04	add				
4	sub $4, $1, $2	↑ 2	04→08	sub	add			
8	or $5, $1, $2	↑ 3	08→12	or	sub	add		
12	and $6, $1, $2	↑ 4	12→16	and	or	sub	add →	
16	xor $7, $1, $2	↑ 5	16→20	xor	and	or	sub	add

- ✗ add-sub：sub第2周期时要读取$1，但add在第5周期才完成回写结果
- ✗ add-or： or第3周期时要读取$1，但add在第5周期才完成回写结果
- ? add-and：两条指令在同一周期访问$1（一个读一个写），and能否读出正确结果？
- ✓ add-xor：add第5个↑写入结果，xor第5周期(第5个↑之后)读取结果，因此执行正确

事实1：计算结果在回写前已经产生了

- 新的计算结果：先出现在流水线寄存器，然后才会保存到RF
- 示例：add进入D级后
  - 计算结果需要4个周期写入寄存器堆
  - 但2个周期后，计算结果已经出现在M级了

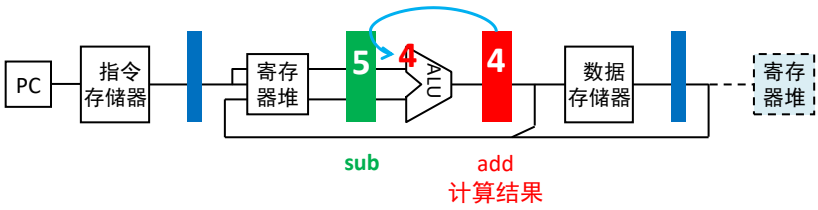
```
地址        指令
 0   add $1, $2, $3
 4   sub $4, $1, $2
 8   or  $5, $1, $2
12   and $6, $1, $2
16   xor $7, $1, $2
```



事实2：功能部件是寄存器值的最终使用者

- 如何理解sub的正确性？
  - 示例：假设RF中的$1值为5，add的计算结果为4
- 读取寄存器值是手段，目的是服务于后续的使用
- 当sub进入E后，只要ALU使用的是M级的结果而非E级的值，则正确性就能保证

```
地址        指令
0   add $1, $2, $3
4   sub $4, $1, $2
8   or  $5, $1, $2
12  and $6, $1, $2
16  xor $7, $1, $2
```



55

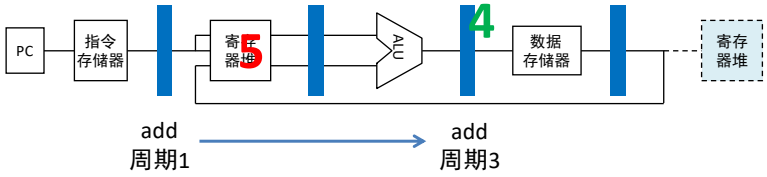
从计算结果产生角度观察1/3

- 示例：周期1~add进入流水线
  - 2个时钟周期后，ALU的计算结果(4)将保存在M级寄存器（意味着add进入M级寄存器）

```
地址        指令
0   add $1, $2, $3
```

		RF(读)	ALU	DM		
CLK	PC	IM	D	E	M	W	RF
↑1	0	add	add				
	4		写$1 2				

2个周期后，$1的最新值将产生



※假设：$1的值为5，add的计算结果为4

56

从功能部件使用寄存器值角度观察2/3

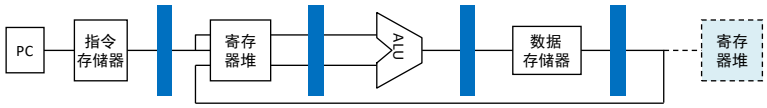
- 示例：周期2~add进入E级
  - 经过1个时钟周期后，结果产生的时间减少1个周期

地址 指令

0 add $1, $2, $3

4 sub $4, $1, $2

			RF(读)	ALU	DM		
CLK	PC	IM	D	E	M	W	RF
↑1	0	add	add				
	4	sub	写$1 2				
↑2	4	sub		add			
	8	or		写$1 1			



add

从功能部件使用寄存器值角度观察3/3

- 示例：周期2~sub进入流水线
  - 当sub进入D级后，其将在下个周期（周期3）进入E级
  - 从ALU角度，意味着再经过1个周期就要用$1的值

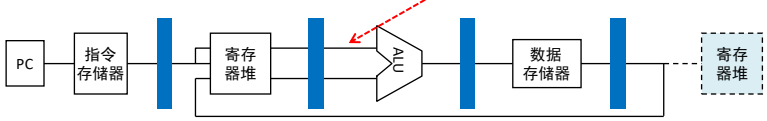
地址 指令

0 add $1, $2, $3

4 sub $4, $1, $2

			RF(读)	ALU	DM		
CLK	PC	IM	D	E	M	W	RF
↑1	0	add	add				
	4	sub	写$1 2				
↑2	4	sub	sub	add			
	8	or	读$1 1	写$1 1			

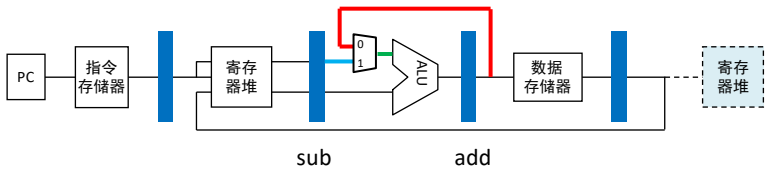
1个周期后：sub进入E，ALU要用$1值



sub add

数据冒险解决策略—转发1/2

- 转发（或旁路）：将尚未写入RF但已经暂存在流水线寄存器中的计算结果传递给相关功能部件(或流水线寄存器)的技术
- 示例：M级向ALU的A输入端的转发
  - 这个转发电路就能很好的解决add-sub间的数据冒险

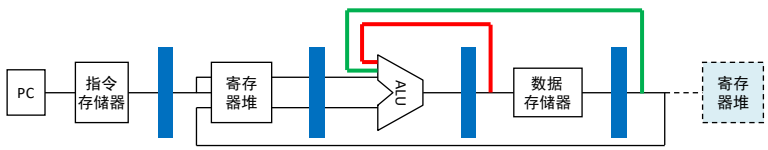


- 转发的2个基本要点
  - 前递：将M级(后级)保存的计算结果向E级(前级)的ALU(功能单元)传递
  - 选择：如果E级指令(sub)读的寄存器与M级指令(add)写的寄存器是同一寄存器时，则选择转发的计算结果为操作数，否则选择E级传递的寄存器值

59

数据冒险解决策略—转发2/2

- 在设计转发时，要考虑完备性
- 示例：ALU的A输入端
  - ALU处于E级，因此M级和W级均可能保存有前序指令的计算结果
  - 只有将M级和W级均转发到E级，才确保了相关组合指令的正确执行



- M→E转发：能够解决的相关例子

```
add $1, $2, $3
sub $4, $1, $6
```

- W→E转发：能够解决的相关例子

```
add $1, $2, $3
XXX
or  $5, $1, $2
```

60

Load导致的数据冒险

- 周期1：lw进入流水线
  - 3个周期后，W级将保存lw从DM中读出的数据

地址	指令	CLK
0	lw $t0, 0($t1)	↑1
4	sub $t3, $t0, $t2	↑2
8	and $t5, $t0, $t4	↑3
12	or $t7, $t0, $t6	↑4
16	add $t1, $t2, $t3	↑5

		RF(读)	ALU	DM		
PC	IM	D	E	M	W	RF
0	lw	lw				
4	sub	写t0 3				

Load导致的数据冒险

- 周期2：sub进入流水线
  - sub再过1个周期到达E时，ALU要用$t0
  - 而lw还需要2个周期才能产生结果

地址	指令	CLK
0	lw $t0, 0($t1)	↑1
4	sub $t3, $t0, $t2	↑2
8	and $t5, $t0, $t4	↑3
12	or $t7, $t0, $t6	↑4
16	add $t1, $t2, $t3	↑5

		RF(读)	ALU	DM		
PC	IM	D	E	M	W	RF
0	lw	lw				
4	sub	写t0 3				
4	sub	sub	lw			
8	and	读t0 1	写t0 2			

- 结论：新值产生太晚！除暂停sub，无任何办法能消除这个冲突

Load导致的数据冒险

- 周期3：插入NOP
  - 所谓的NOP，就是清空E级，相当于插入空拍
  - 注意，除了清空E级，还需要冻结PC（即PC必须保持不变）

地址	指令	CLK	PC	IM	D	E	M	W	RF
					RF(读)	ALU	DM		
0	lw $t0, 0($t1)	↑1	0 4	lw sub	lw 写t0 3				
4	sub $t3, $t0, $t2	↑2	4 8	sub and	sub 读t0 1	lw 写t0 2			
8	and $t5, $t0, $t4	↑3	8 8	and and	sub 读t0 1	nop	lw 写t0 1		
12	or $t7, $t0, $t6	↑4							
16	add $t1, $t2, $t3	↑5							

- 由于插入了NOP，因此冲突解除；转发机制将在周期4发挥作用

63

Load导致的数据冒险

- 周期4-1：sub到达E
  - lw：从DM中读出的数据($t0新值)被写入W
  - sub：选择来自W级转发的计算结果

地址	指令	CLK	PC	IM	D	E	M	W	RF
					RF(读)	ALU	DM		
0	lw $t0, 0($t1)	↑1	0 4	lw sub	lw 写t0 3			计算结果	
4	sub $t3, $t0, $t2	↑2	4 8	sub and	sub 读t0 1	lw 写t0 2			
8	and $t5, $t0, $t4	↑3	8 8	and and	sub 读t0 1	nop	lw 写t0 1		
12	or $t7, $t0, $t6	↑4	8 12			sub 读t0 0	nop	lw 新t0 0	
16	add $t1, $t2, $t3	↑5							

64

Load导致的数据冒险

- 周期4-2：and进入流水线
  - lw-xxx-and同样需要转发，用W级计算结果替代and读出的$1
  - 注意：这个转发的目的地不是功能单元，而是E级流水线寄存器

地址	指令	CLK	PC	IM	D	E	M	W	RF
					RF(读)	ALU	DM		
0	lw $t0, 0($t1)	↑1	0 4	lw sub	lw 写t0 3			计算结果	
4	sub $t3, $t0, $t2	↑2	4 8	sub and	sub 读t0 1	lw 写t0 2			
8	and $t5, $t0, $t4	↑3	8 8	and and	sub 读t0 1	nop	lw 写t0 1		
12	or $t7, $t0, $t6	↑4	8 12	and or	and 读t0 1	sub 读t0 0	nop	lw 新t0 0	
16	add $t1, $t2, $t3	↑5							

65

Load导致的数据冒险

- 周期5：or进入流水线
  - 由于流水线中没有写$t0的指令了，这意味着RF中的$t0是最新值
  - or指令可以安全的从RF中读取$t0

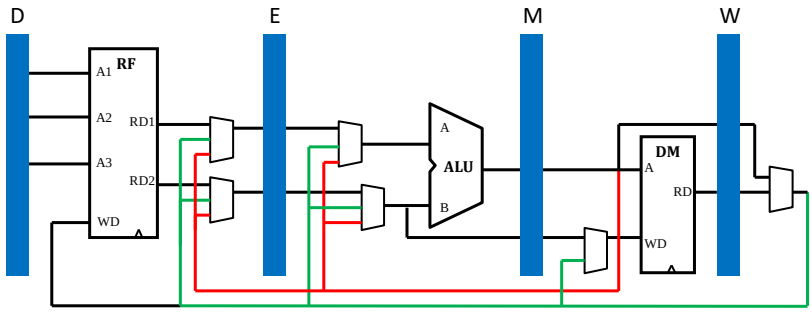
地址	指令	CLK	PC	IM	D	E	M	W	RF
					RF(读)	ALU	DM		
0	lw $t0, 0($t1)	↑1	0 4	lw sub	lw 写t0 3				
4	sub $t3, $t0, $t2	↑2	4 8	sub and	sub 读t0 1	lw 写t0 2			
8	and $t5, $t0, $t4	↑3	8 8	and and	sub 读t0 1	nop	lw 写t0 1		
12	or $t7, $t0, $t6	↑4	8 12	and or	and 读t0 1	sub 读t0 0	nop	lw 写t0 0	
16	add $t1, $t2, $t3	↑5	12 16	or add	or 读t0 1	and	sub	nop	更新t0

66

详细的转发电路及其控制

- 最新结果：可能出现在M和W
- 使用结果：D、E、M各级可能的位置
  - D级：RS寄存器值、RT寄存器值
  - E级：ALU的A、B；RT寄存器值
  - M级：DM的WD

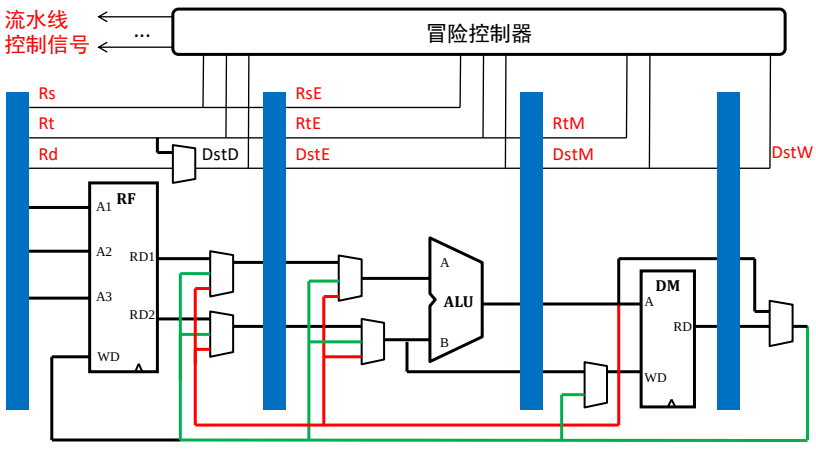
	M级ALU计算结果	W级回写结果
D级被转发	RS寄存器值 RT寄存器值	RS寄存器值 RT寄存器值
E级被转发	ALU的A ALU的B RT寄存器值	ALU的A ALU的B RT寄存器值
M级被转发		DM的WD



67

冒险控制器

- 基本功能：检测和分析各类冒险，并控制流水线的执行
- 先讨论应对数据冒险的基本控制方法



68

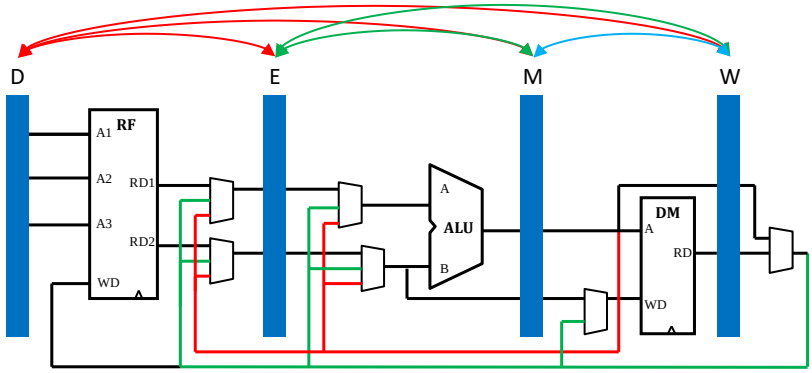
数据冒险的检测与控制

- 基本策略

D级　分析与E/M/W的相关性。若转发无法解决则暂停，直至转发可以解决相关

E级　E级已无暂停问题，只需要分析与M/W的相关性，决定是否转发即可

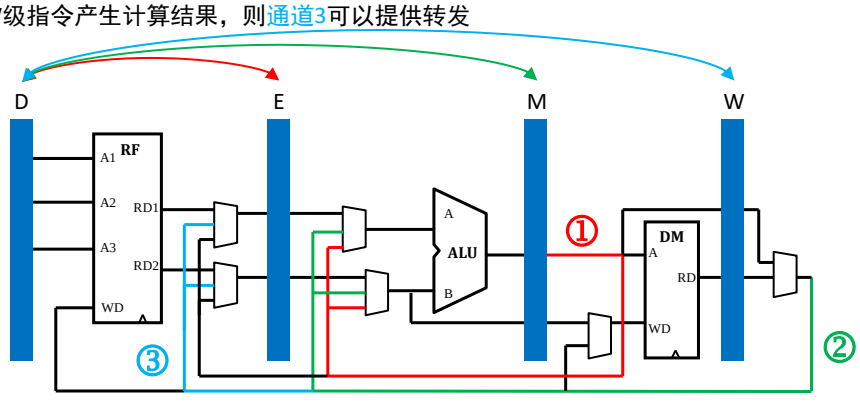
M级　M级已无暂停问题，只需要分析与W的相关性，决定是否转发即可



69

数据冒险的检测与控制：暂停(以RS为例)1/10

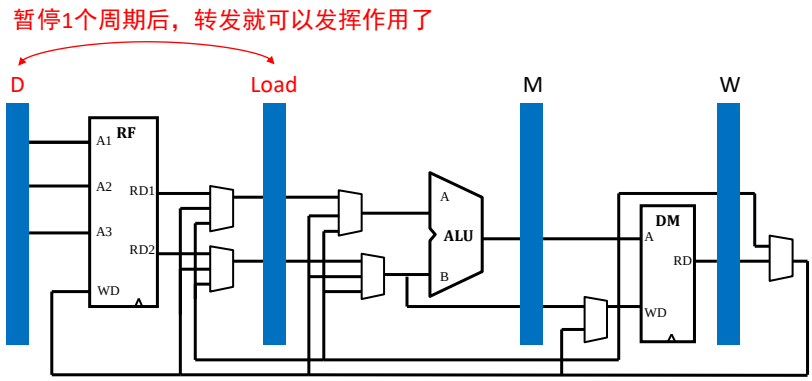
- 对于D级指令来说，哪些前序指令会导致暂停？
  - 类别1：计算类指令（如add，addi）
    - 如果由E级指令产生计算结果，则通道①可以提供转发
    - 如果由M级指令产生计算结果，则通道②可以提供转发
    - 如果由W级指令产生计算结果，则通道3可以提供转发



70

数据冒险的检测与控制：暂停(以RS为例)2/10

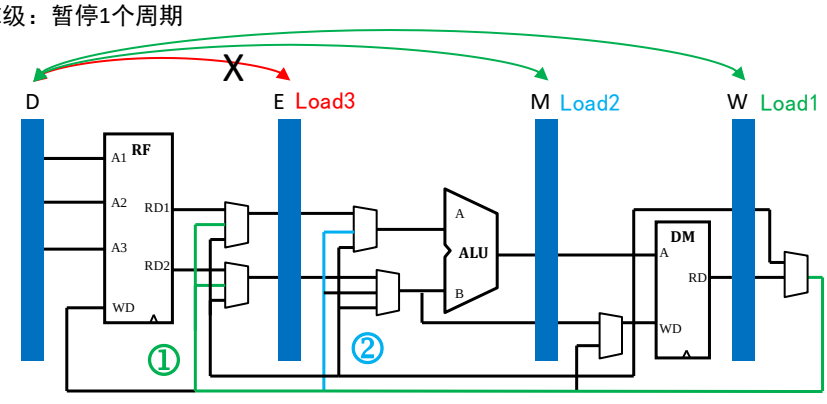
- 对于D级指令来说，哪些前序指令会导致暂停？
  - 类别2：Load指令（如lw/lh/lhu/lb/lbu）
    - Load在E级：当D级指令进入E时，Load进入M，结果尚未产生，只能暂停



71

数据冒险的检测与控制：暂停(以RS为例)3/10

- 对于D级指令来说，哪些前序指令会导致暂停？
  - 类别2：Load指令（如lw/lh/lhu/lb/lbu）
    - Load在W级：由于结果已经在W级了，因此通道①转发
    - Load在M级：当D级指令进入E时，Load的结果存入W，通道②转发
    - Load在E级：暂停1个周期



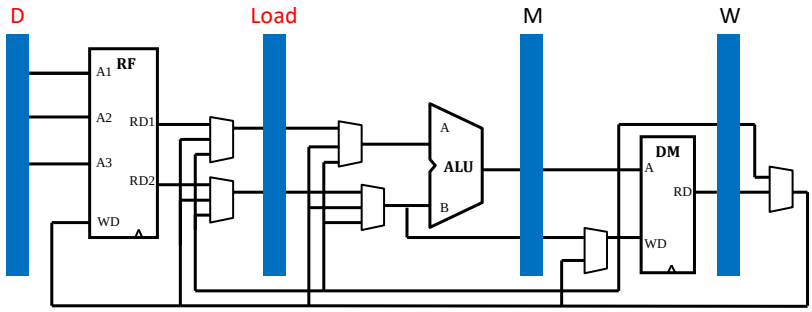
72

数据冒险的检测与控制：暂停(以RS为例)4/10

- Load导致的暂停有哪些条件？
  - 条件1：D级为要读寄存器的指令
  - 条件2：E级为load类指令
  - 条件3：读的寄存器 == 写的寄存器

Q1：哪些指令读寄存器？

Q2：哪些指令是load类指令？



73

数据冒险的检测与控制：暂停(以RS为例)5/10

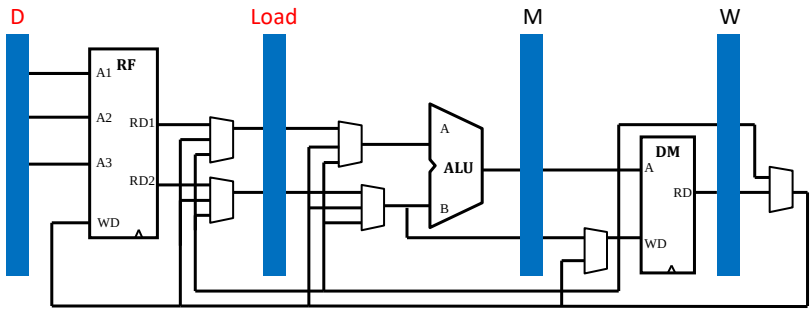
- 暂停的检测：表达式

```
Stall = (add + sub +…+ lw) & lwE & (A1==A3E)
```

寄存器编号相同

E级为Load类

D级会读rs寄存器的指令

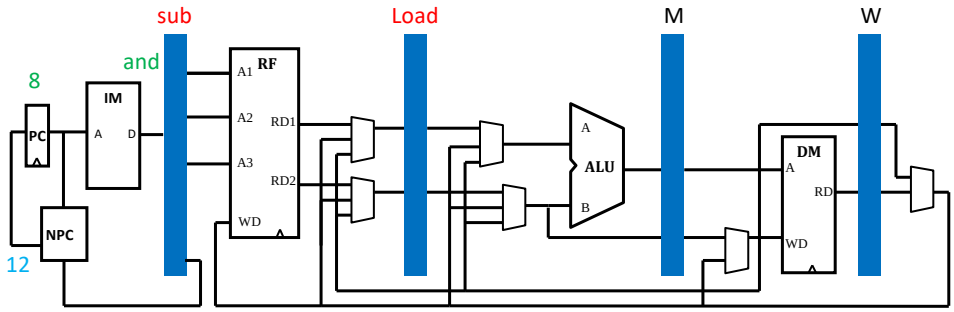


74

数据冒险的检测与控制：暂停(以RS为例)6/10

- 暂停的执行动作
  - ①冻结D：sub停留在D，不得前进
  - ②清除E：E级整体清0，等价于插入NOP
  - ③禁止PC：防止PC继续计数，PC应保持为PC+4

```
地址        指令
 0  lw   $t0, 0($t1)
 4  sub  $t3, $t0, $t2
 8  and  $t5, $t0, $t4
12  or   $t7, $t0, $t6
16  add  $t1, $t2, $t3
```

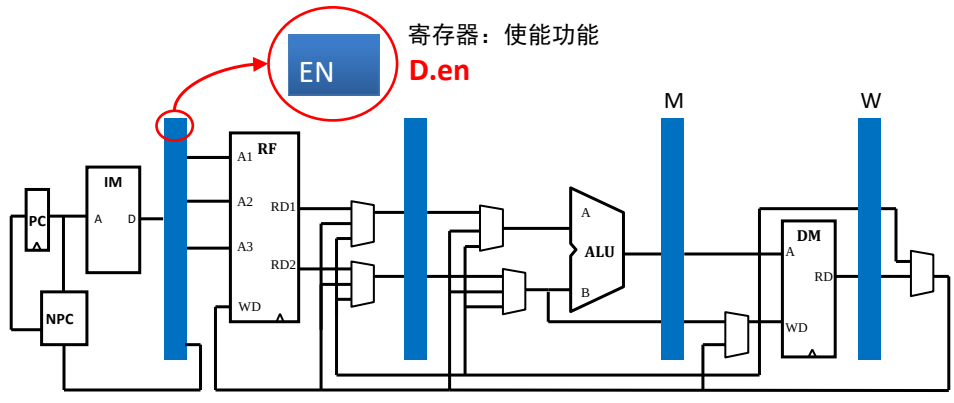


75

数据冒险的检测与控制：暂停(以RS为例)7/10

- 暂停的执行动作
  - ①冻结D：sub停留在D，不得前进
  - ②清除E：E级整体清0，等价于插入NOP
  - ③禁止PC：防止PC继续计数，PC应保持为PC+4

数据通路	将D寄存器修改为使能型寄存器
控制器	增加D寄存器使能信号D.en

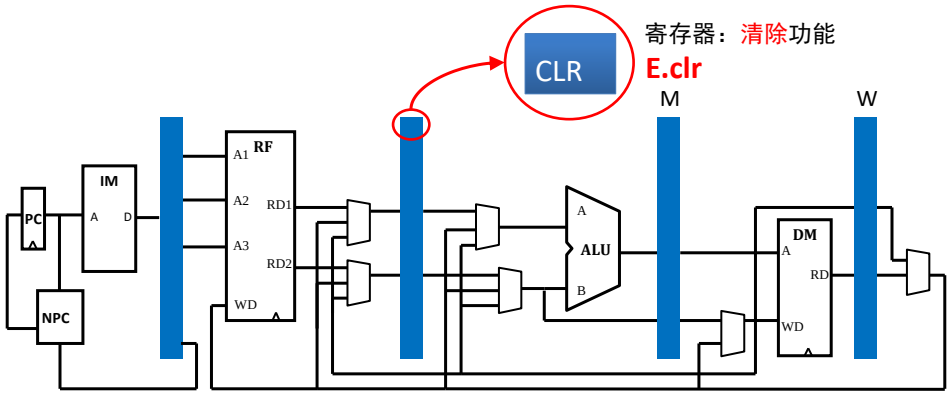


76

数据冒险的检测与控制：暂停(以RS为例)8/10

- 暂停的执行动作
  - ①冻结D：sub停留在D，不得前进
  - ②清除E：E级整体清0，等价于插入NOP
  - ③禁止PC：防止PC继续计数，PC应保持为PC+4

数据通路	将E寄存器修改为复位型寄存器
控制器	增加E寄存器清除信号E.clr

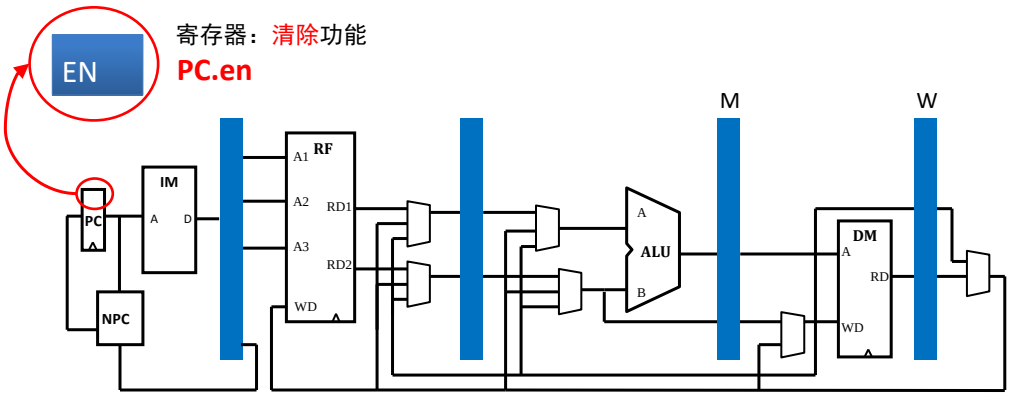


77

数据冒险的检测与控制：暂停(以RS为例)9/10

- 暂停的执行动作
  - ①冻结D：sub停留在D，不得前进
  - ②清除E：E级整体清0，等价于插入NOP
  - ③禁止PC：防止PC继续计数，PC应保持为PC+4

数据通路	将PC修改为使能型寄存器
控制器	增加PC使能信号PC.en

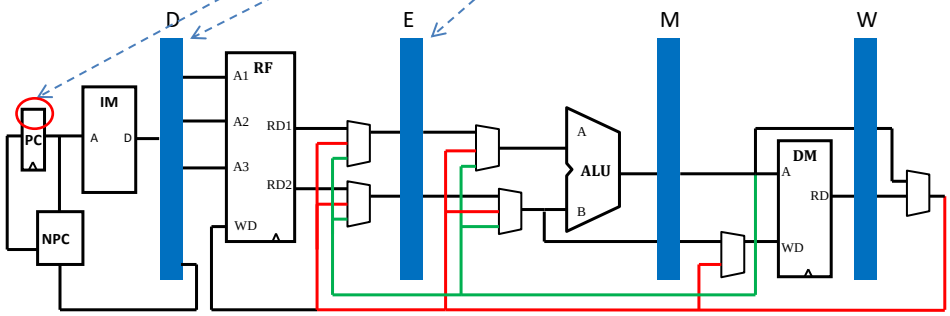


78

数据冒险的检测与控制：暂停(以RS为例)10/10

- 暂停的执行动作：控制信号表达式
  - ①冻结D：sub停留在D，不得前进
  - ②清除E：E级整体清0，等价于插入NOP
  - ③禁止PC：防止PC继续计数，PC应保持为PC+4

```
PC.en = !Stall
D.en  = !Stall
E.clr =  Stall
```



79

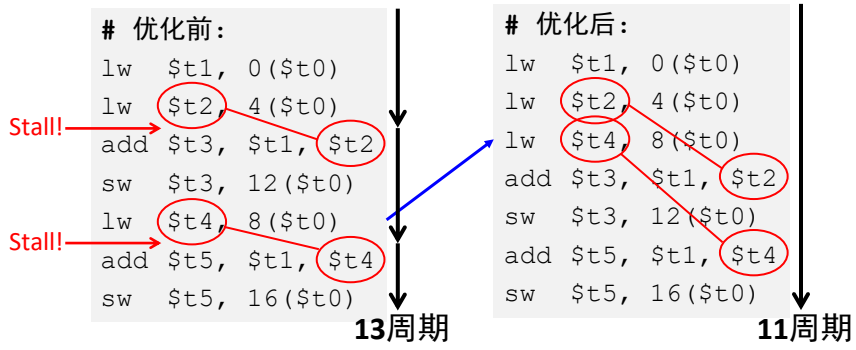
编译优化

- 如果load的后继指令与load相关，则需要暂停1个周期
  - 这个暂停周期被称为加载延迟槽（load delay slot）
- 如果将与load无关的指令放置在load类指令后，则就不需要暂停了
  - 这就是编译优化
- 调度的指令，不仅要考虑load无关性，还需要确保程序员视角的正确性

80

编译优化

- 编译优化：重排序指令序列从而避免load类相关
- C代码：**A=B+E； C=B+F；**



81

目录

- 流水线概述
- 流水线数据通路
- 流水线控制
- 流水线冒险
  - 控制冒险
- 流水线性能分析
- 2种CPU模型对比

82

控制冒险

- 分支指令（beq，bne）影响控制流
  - 顺序取指还是转移取指取决于ALU的比较结果
  - 在比较结果产生以前，D级取指无法确保是正确的
- 简单方案：暂停分支指令直至产生正确的PC值
- 新的问题：需要等待多长时间？

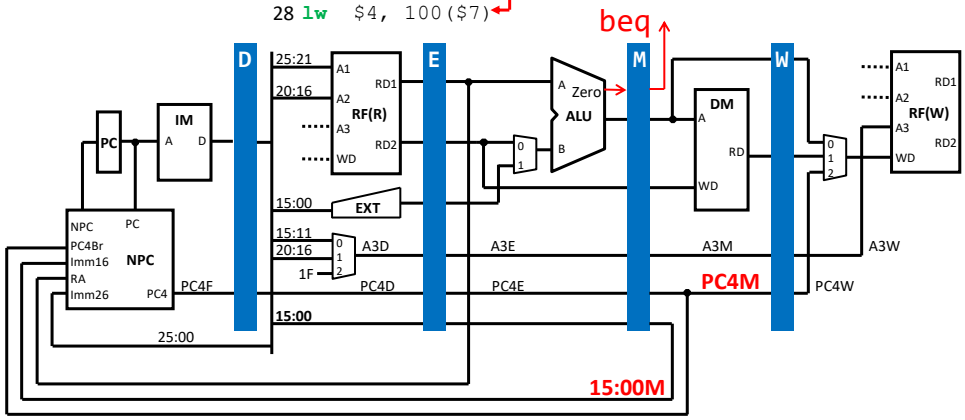
B指令冒险造成的停顿代价

- 简单暂停导致3个NOP
  - Zero、PC4、偏移均在M级，故PC在clk4才能加载正确值
  - D级在clk5才能存入转移指令(即lw)

```
地址        指令
 0 beq  $1,  $3, 24
 4 and  $12, $2, $5
 8 or   $13, $6, $2
12 add  $14, $2, $2

28 lw   $4, 100($7)
```

CLK	RF(读) PC	ALU IM	D	E	M	W	DM RF
↑1	0→4	beq→and	beq				
↑2	4→4	and→and	nop(1)	beq			
↑3	4→4	and→and	nop(2)	nop(1)	beq		
↑4	4→28	and→lw	nop(3)	nop(2)	nop(1)		
↑5	28→32	lw→XXX	lw	nop(3)	nop(2)	nop(1)	



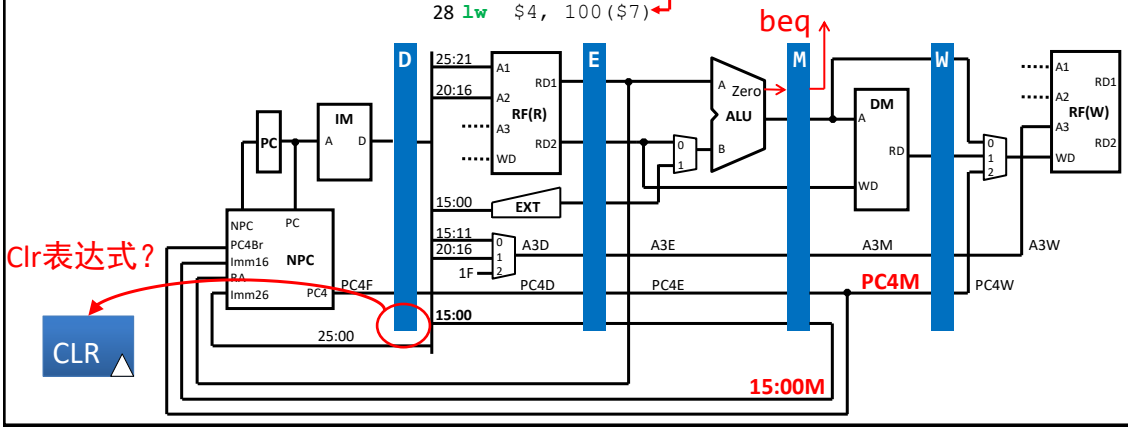
B指令冒险造成的停顿代价

- 简单暂停导致3个NOP
  - Zero、PC4、偏移均在M级，故PC在clk4才能加载正确值
  - D级在clk5才能存入转移指令(即lw)

```
地址        指令
 0 beq  $1,  $3, 24
 4 and  $12, $2, $5
 8 or   $13, $6, $2
12 add  $14, $2, $2

28 lw   $4, 100($7)
```

CLK	PC	IM	D	E	M	W	RF
			RF(读)	ALU	DM		
1	0→4	beq→and	beq				
2	4→4	and→and	nop(1)	beq			
3	4→4	and→and	nop(2)	nop(1)	beq		
4	4→28	and→lw	nop(3)	nop(2)	nop(1)		
5	28→32	lw→XXX	lw	nop(3)	nop(2)	nop(1)	

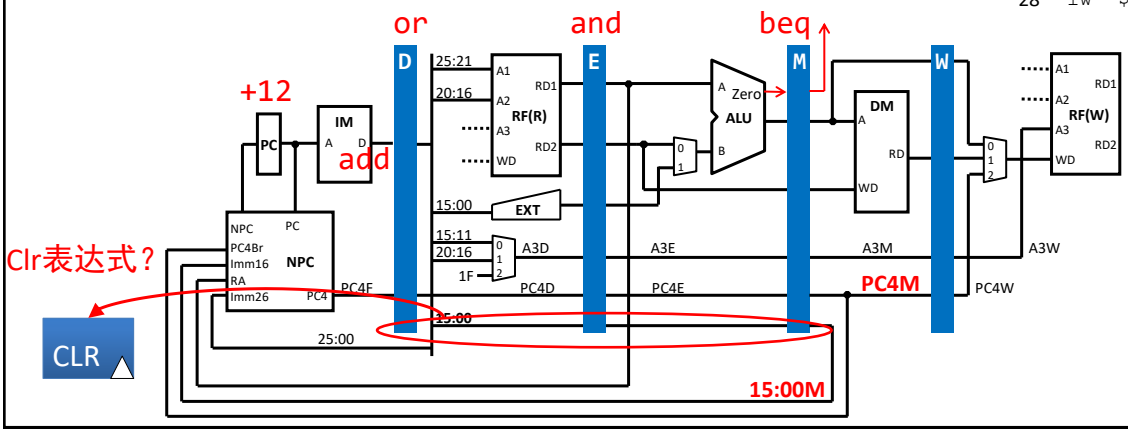


方案1：假定分支不发生

- 措施1：即使在D级发现是B指令也不停顿
- 措施2：然后根据B指令结果，决定是否清除3条后继指令
  - 清除，即使得and/or/add不能前进

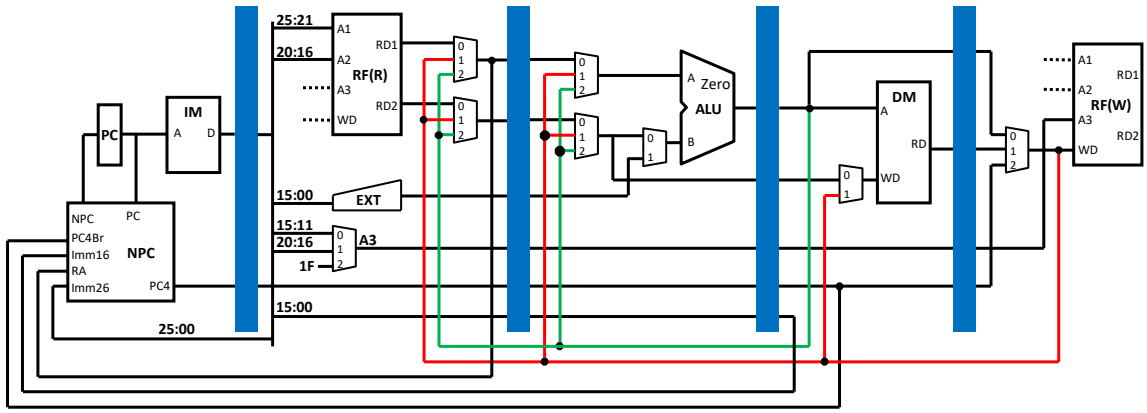
```
地址        指令
 0  beq  $1,  $3, 24
 4  and  $12, $2, $5
 8  or   $13, $6, $2
12  add  $14, $2, $2

28  lw   $4, 50($7)
```



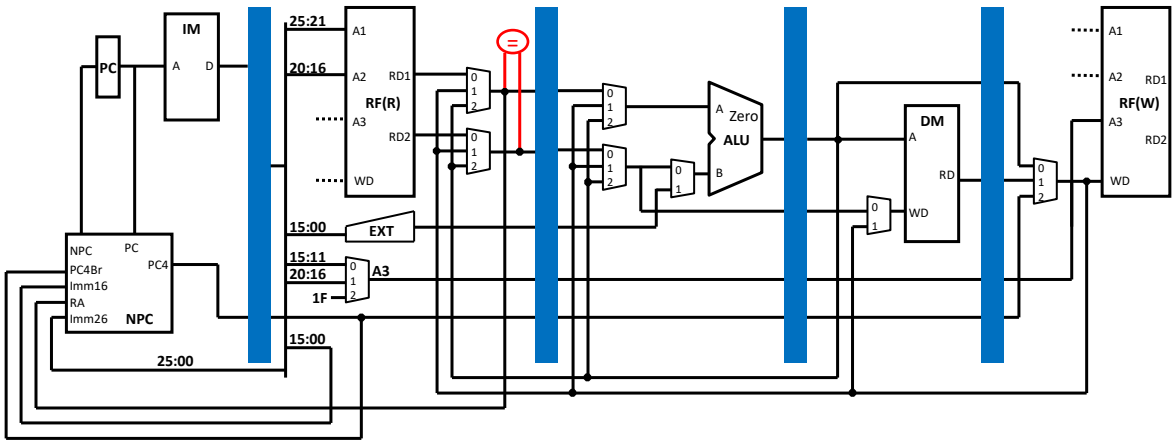
完整流水数据通路

- 1）beq：在ALU处
- 2）jal：RS在E级



完整版数据通路：beq前移

- Beq前移



方案2：缩短分支延迟1/3

- 在D级放置比较器，尽快得到B指令结果
  - B指令结果可以提前2个clock得到
  - B指令后继可能被废弃的指令减少为1条
    - 当需要转移时，清除D级即可

```
地址        指令
 0 beq  $1,  $3, 24
 4 and  $12, $2, $5
 8 or   $13, $6, $2
12 add  $14, $2, $2

28 lw   $4, 100($7)
```

CLK	PC	IM	D	E	M	W	RF
			RF(读)	ALU	DM		
1	0→4	beq→and	beq				
2	4→4	and→and	nop(1)	beq			
3	4→4	and→and	nop(2)	nop(1)	beq		
4	4→28	and→lw	nop(3)	nop(2)	nop(1)		
5	28→32	lw→XXX	lw	nop(3)	nop(2)	nop(1)	

比较器在ALU

```
地址        指令
 0 beq  $1,  $3, 24
 4 and  $12, $2, $5
 8 or   $13, $6, $2
12 add  $14, $2, $2

28 lw   $4, 100($7)
```

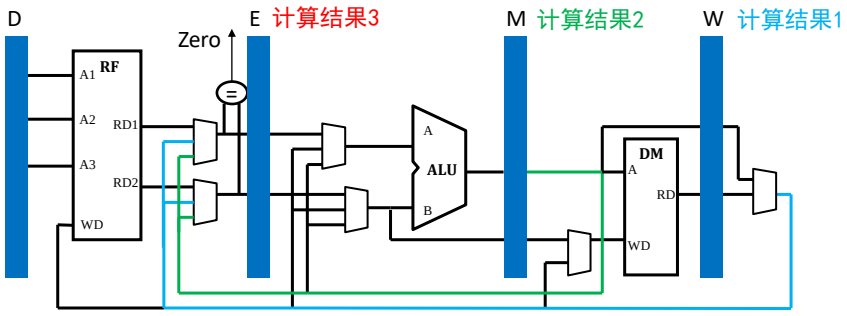
CLK	PC	IM	D	E	M	W	RF
			RF(读)	ALU	DM		
1	0→4	beq→and	beq				
2	4→28	and→lw	and				
3	28→32	lw→XXX	lw				

比较器前置

89

方案2：缩短分支延迟2/3

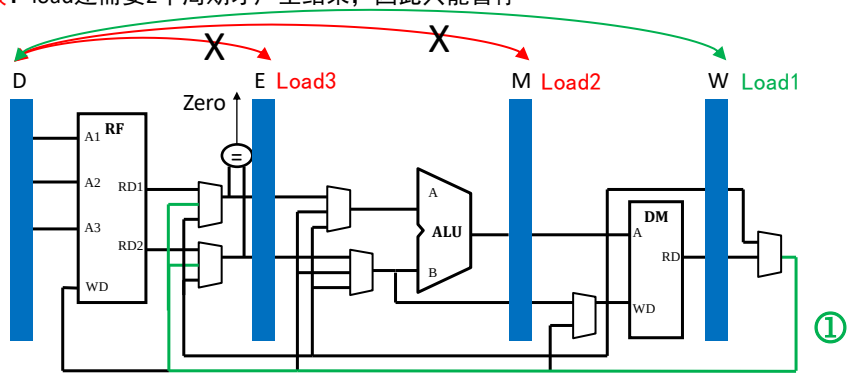
- 比较器前置后会产生数据相关（可能依赖于前序指令的结果）
  - 类别1：如果前序指令中有计算类指令（如add，addi）
    - 依赖W级指令的计算结果1：从W转发数据
    - 依赖M级指令的计算结果2：从M转发数据
    - 依赖E级指令的计算结果3：只能暂停



90

方案2：缩短分支延迟3/3

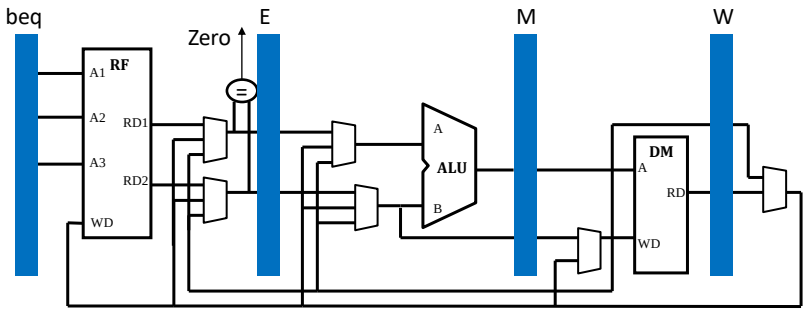
- 比较器前置后会产生数据相关（可能依赖于前条指令的结果）
  - 类别2：如果前序指令中有load类指令
    - Load在W级：结果已经产生，转发！
    - Load在M级： load还需要1个周期才产生结果，因此只能暂停
    - Load在E级：load还需要2个周期才产生结果，因此只能暂停



91

深入思考

- Q1：除了计算类指令、load类指令，是否还有其他指令会产生寄存器结果？
- Q2：如果存在这样的指令，寄存器新值最早会出现在哪级寄存器？
- Q3：对于Q2来说，能否构造出会与beq数据相关的指令序列？



92

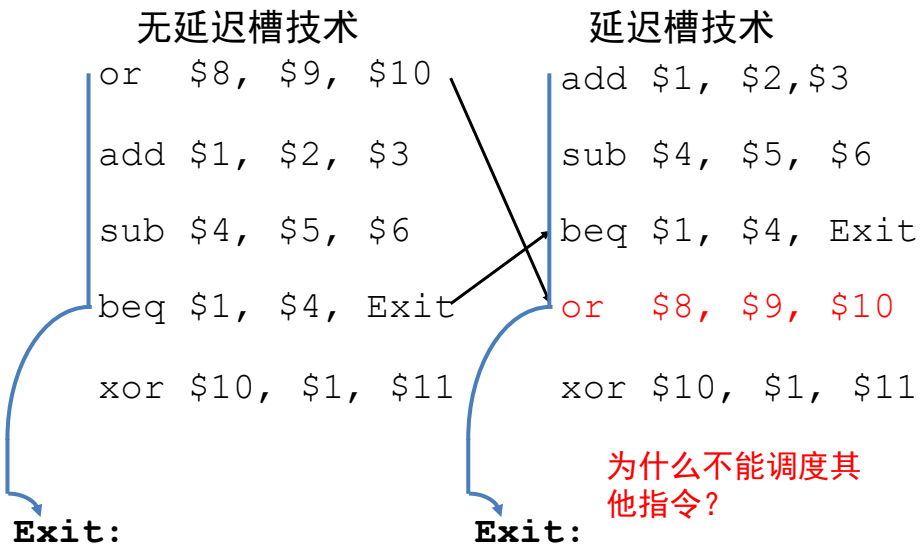
方案3：分支延迟槽1/3

- 无论是否执行转移方向的指令，都执行分支指令后面的那条指令
  - 这一技术就是分支延迟槽（Branch delay slot）
- 简单方案：在分支指令后面放置一条nop
- 优化方案：编译从分支指令前面的指令中调度一条无关指令到分支指令后面
  - 编译优化必须确保不能改变程序逻辑
- J、Jal、Jr：是否也可以采用这一技术？

93

方案3：分支延迟槽2/3

- 示例



94

方案3：分支延迟槽3/3

- 分支指令的PC+8问题
  - 常规上，分支指令是以PC+4为基地址加偏移来计算转移地址
  - 由于在分支指令后面人为的放一条指令，因此转移地址就多了4
  - 支持延迟槽的CPU需要自动用PC+8作为基地址
- Jal指令的PC+8问题
  - 对于jal，可以同样在其后人为的放一条指令。同样，返回地址多了4
  - 为此，保存在R[31]的地址就不能是PC+4，而必须是PC+8！
    - 否则`jr $31`会返回至延迟槽指令，而不是延迟槽下面的那条指令

95

目录

96

流水线冒险小结

- 冒险降低流水线效率
  - 导致暂停
- 结构冒险
  - 由于功能部件争用所致
- 数据冒险
  - 需要等待前序指令的执行结果
- 控制冒险
  - 控制流的方向确定前，下条指令无法确定
  - 分支指令和跳转指令延迟槽

97

目录

- 流水线概述
- 流水线数据通路
- 流水线控制
- 流水线冒险
- 流水线性能分析
- 2种CPU模型对比

98

流水线CPI

- 理论上，流水线每个时钟周期能执行完1条指令，即

$$CPI_{理想}=1$$

- 然而，冒险会导致流水线暂停，即

$$CPI_{实际}>1$$

- 因为不同的冒险带来不同的暂停代价，所以$CPI_{实际}$与运行程序各类指令的占比相关

99

流水线CPI

- 示例：计算流水线的CPI
  - 某程序指令分布如下：load占25%，store为10%，分支指令为11%，R型计算类指令为54%
  - 假设：①load导致暂停概率为40%，且暂停代价为1个时钟周期。②分支指令预测成功率为75%，但预测失败就需要暂停1个时钟周期
- 解
  - load：无数据相关时，load的CPI为1；有数据相关时，CPI为2

$$CPI_{load}=1\times(1-40\%)+2\times40=1.4$$

  - store：CPI_{store}为1
  - 分支：预测成功，分支的CPI为1；预测失败，分支的CPI为2

$$CPI_{分支}=1\times75\%+2\times(1-75\%)=1.25$$

  - R型：$CPI_{R型}$为1

$$CPI=CPI_{load}\times25\%+CPI_{store}\times10\%+CPI_{分支}\times11\%+CPI_{R型}\times54\%=1.1275$$

100

目录

101

2种CPU模型对比

- 执行周期
  - 单周期：恒为1
  - 流水线：理想为1，但冒险存在导致大于1
- 时钟频率
  - 单周期：关键路径存在导致最低
  - 流水线：分段有利于提升频率；转发和控制导致频率下降

102

2种CPU模型对比

- 性能计算
  - 单周期：仅与执行的指令数及时钟周期宽度相关
  - 流水线：不仅与时钟周期宽度相关，还与指令频度及其相应的CPI相关
- 设计要点
  - 单周期：数据通路是基础；控制信号仅与指令功能相关
  - 流水线：数据通路通过切割关键路径来提升时钟频率，还通过转发解决数据冒险；控制信号分为主控制与冒险控制两大部分
    - 主控制：与单周期、多周期相同，只与单条指令相关
    - 冒险控制：主要解决各类冒险带来的转发与暂停